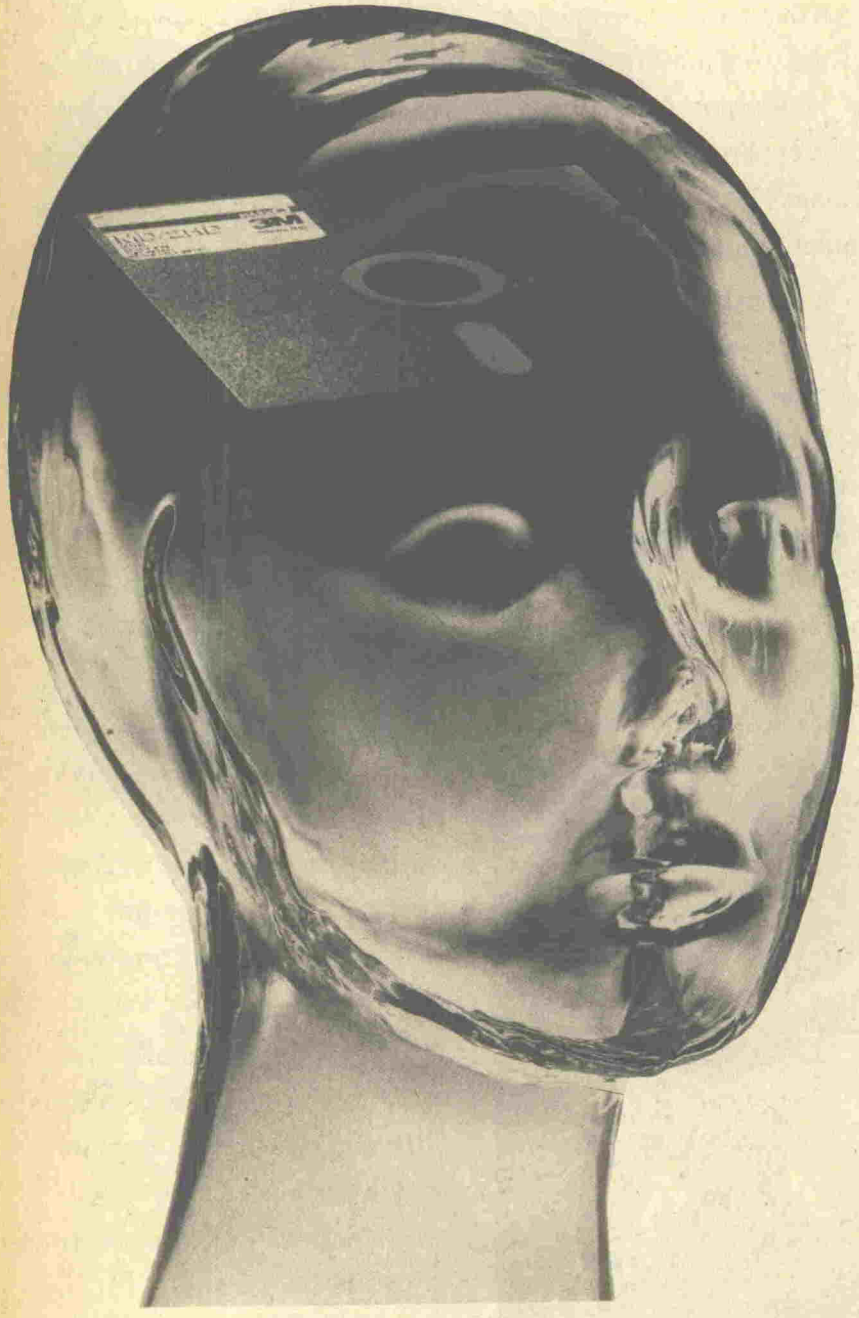
โปรแกรมในยุคที่ สาม



บุญเลิศ เอี่ยมทัศนา

การแบ่งยุคสมัยของคอมพิวเตอร์แบ่งกันได้หลายด้าน เช่น เมื่อแบ่งทางด้านฮาร์ดแวร์ ยุคแรกจะเป็นยุคของหลอดและรีเลย์ ยุคที่สองเป็นยุคของทรานซิสเตอร์ ยุคที่สามเป็นยุคของไอซี ยุคที่สี่คือยุคปัจจุบันเป็นยุคของวีแอลเอสไอ ส่วนยุคที่ห้าคือยุคของอนาคตนั้น คาดกันว่าจะมีการเปลี่ยนแปลงโครงสร้างภายในของเครื่องคอมพิวเตอร์ไปจากเดิมโดยสิ้นเชิง

ทางด้านภาษา ยุคแรกเป็นยุคของภาษาเครื่อง ยุคที่สองเป็นยุคของภาษาสัญลักษณ์ คือภาษาที่รู้จักกันโดยทั่วไปในปัจจุบัน ยุคที่สามเป็นยุคของภาษาที่ดำเนินการกับฐานข้อมูล เช่น ดีเบส ยุคที่สี่เป็นยุคที่ผมเองยังไม่เข้าใจชัดเจนนัก คล้ายกับว่าเป็นภาษาที่ใช้สำหรับเขียนโปรแกรมอีกทอดหนึ่ง โดยที่ผู้เขียนโปรแกรมเพียงแต่แถลงความต้องการ โปรแกรมตัวแปรภาษาในยุคที่สี่จะแปลเป็นโปรแกรมให้

เมื่อพิจารณาในด้านการเขียนโปรแกรมบ้าง ผมเห็นว่าน่าจะแบ่งเป็นยุคสมัยได้เช่นกัน คือ ยุคแรกเป็นการเขียนโปรแกรมรวมอยู่กับข้อมูล ซึ่งเป็นยุคเดียวกับภาษาเครื่อง ดังจะเห็นร่องรอยอยู่ในภาษาเบสิก ยุคที่สองเป็นการเขียนโปรแกรมแยกจากข้อมูล ดังเช่นโปรแกรมทั่วๆไปในปัจจุบัน ส่วนยุคที่สามเป็นการเขียนโปรแกรมให้ผู้ใช้เขียนโปรแกรมซ้อนโปรแกรมได้ดังเช่นดีเบส (เมื่อมองในฐานะโปรแกรม) หรือให้ผู้ใช้กำหนดสูตรในการคำนวณได้ เช่นโปรแกรมพวกสเปรดชีต

จะเห็นว่าการเขียนโปรแกรมในยุคที่สามนี้ต้องการความรู้นอกเหนือไปจากที่ศึกษากันมาในการเขียนโปรแกรมตามแบบ ซึ่งการเขียนโปรแกรมเพื่ออ่านโปรแกรมของผู้ใช้มาตีความ (ดังในดีเบส) ได้กล่าวถึงแล้วในเรื่องภาษาไอคิวซีโร ในบทความนี้จะกล่าวถึงการเขียนโปรแกรมเพื่อให้ผู้ใช้ป้อนสูตรในการคำนวณแล้วนำมาคำนวณ ดังเช่นที่ใช้ในพวกสเปรดชีต แต่จะขอกล่าวตรงจุด

เริ่มต้นเท่านั้น และถึงแม้โปรแกรมตัวอย่างในเรื่องนี้จะเขียนด้วยภาษาปาสคาล แต่เนื้อหาใช้ได้กับทุกภาษา

ถ้อยแถลงให้ค่า (assignment statement)

ไม่ว่าจะเป็นฟังก์ชันทางคณิตศาสตร์ หรือสูตรในการคำนวณ หรือการดำเนินกรรมวิธีในการคำนวณใดๆ เพื่อให้ค่าแก่ตัวแปร เช่น

$y = x^2+2x+5$

$y = a+r^2$

$z = 5+4\times3-8/2$

$a = 3275.9\times384.5+1279.8756$

ต่างก็เป็นถ้อยแถลงให้ค่าในการเขียนโปรแกรมทั้งสิ้น ถ้อยแถลงเหล่านี้เมื่อนำมาเขียนในโปรแกรมของเราสามารถทำได้โดยง่าย (แต่ก็มีข้อกำหนดในแต่ละภาษาว่าจะดำเนินการอย่างไร) เช่น ถ้อยแถลงที่ให้ค่า z ถ้าเขียนเช่นนี้ในโปรแกรมภาษาปาสคาล จะดำเนินการเป็น

$z = (5+(4\times3))-(8/2) = 13$

ทั้งนี้เพราะในภาษาปาสคาลได้กำหนดไว้ว่า ถ้าถ้อยแถลงใดมีเครื่องหมายในการคำนวณผสมกันอยู่ ให้ทำคูณและหารก่อนบวกและลบ ถ้าเป็นเครื่องหมายในระดับเดียวกัน ให้ทำจากทางซ้ายไปทางขวา ถ้าต้องการให้ทำบวกก่อนต้องเขียนเป็น

$z = (5+4)\times3-8/2 = 23$

เมื่อให้ผู้ใช้ป้อน

แต่ถ้าถ้อยแถลงเหล่านี้ไม่ได้อยู่ในโปรแกรมของเรา หากอยู่ในโปรแกรมที่ให้ผู้ใช้เขียนซ้อน หรืออยู่ในรูป (ซึ่งขอเรียกว่าเป็น) สูตรที่ให้ผู้ใช้ป้อนทั้งบรรทัด จะดำเนินการได้อย่างไร (ขอให้หยุดคิดก่อนครับ)

เรื่องนี้ไม่อยู่ในหัวข้อการเรียนการสอนในการเขียนโปรแกรมธรรมดา หากอยู่ในเรื่องของการเขียนโปรแกรมตัวแปลภาษา หลักการเริ่มต้นคือจะต้องกำหนดเครื่องหมายการคำนวณที่ให้ใช้ได้ว่ามีอะไรบ้าง และมีน้ำหนักเป็นอย่างไร ซึ่งขอกำหนดดังนี้

^ ยกกำลังเลขทางซ้ายด้วยค่ากำลังทางขวา เช่น 3^2 = 9

/ หาร โดยเลขทางซ้ายเป็นตัวตั้งเลขทางขวาเป็นตัวหาร เช่น 12/3 = 4

* คูณ โดยเลขทางซ้ายเป็นตัวตั้งเลขทางขวาเป็นตัวคูณ เช่น 3*2 = 6

– ลบ โดยเลขทางซ้ายเป็นตัวตั้งเลขทางขวาเป็นตัวลบ เช่น 3–2 = 1

+ บวก โดยเลขทางซ้ายเป็นตัวตั้ง เลขทางขวาเป็นตัวบวก เช่น 3+2 = 5

สำหรับลำดับความสำคัญ ให้ทำยกกำลังก่อนการคูณและการหาร การคูณและการหารก่อนการบวกและลบ หากเป็นเครื่องหมายที่อยู่ในระดับเดียวกัน คือ การคูณกับการหาร และการบวกกับการลบ ให้ทำจากซ้ายไปขวาของนิพจน์ (เช่นเดียวกับภาษาปาสคาล) เช่น

x = 4*7 / 2 = 28 / 2 = 14

หากมีวงเล็บให้ถอดวงเล็บคือดำเนินการในวงเล็บก่อน สำหรับเลขยกกำลังซ้อนกัน เช่น 2^3^2 ให้ดำเนินการจากขวามาซ้าย (เพื่อให้ตรงกับการคำนวณ) จะได้ค่าเป็น

2^(3^2) = 2^9 = 512

Infix กับ Postfix[1]

ถ้อยแถลงหรือสูตรดังกล่าวมานี้ ไม่ว่าจะอยู่ในซอร์สโปรแกรม หรือที่ให้ผู้ใช้ป้อน ความจริงแล้วจะอยู่ในรูปของสตริง ไม่ใช่ตัวแปรหรือเลขจำนวนแต่อย่างใด แต่เราจะมองว่าประกอบด้วยตัวแปร เลขจำนวน และเครื่องหมาย

สำหรับการเขียนนิพจน์ตามที่ยกตัวอย่างมาเป็นการเขียนในแบบ Infix คือเขียนในแบบที่นำค่าทางซ้ายกับทางขวามาดำเนินการต่อกัน ซึ่งมนุษย์เข้าใจได้ง่าย แต่ทำให้คอมพิวเตอร์เข้าใจได้ยาก เช่นนิพจน์ว่า

3+4*5

หากเราเขียนโปรแกรมให้ดำเนินการจากซ้ายไปขวา โดยไม่คำนึงถึงน้ำหนักของเครื่องหมาย จะดำเนินการได้โดยตรง แต่เมื่อเรากำหนดน้ำหนักของเครื่องหมาย ซึ่งตามตัวอย่างนี้จะต้องดำเนินการคูณก่อนบวก จะต้องเปลี่ยนนิพจน์นี้เป็นแบบ Postfix ก่อน คือจะเป็น

3 4 5 * +

นิพจน์แบบ Postfix นี้เพื่อการดำเนินการในแบบ นำข้อมูลที่อยู่ทางซ้ายของนิพจน์มา 2 ตัว โดยให้ตัวแรกเป็นตัวตั้งตัวหลังเป็นตัวที่นำมากระทำต่อ เช่น 8 6 – จะได้เป็น 8–6 = 2 สำหรับตัวอย่างที่ยกมาจะเริ่มด้วยการคูณ 4 กับ 5 ได้ค่าเป็น 3 20 + คือ 23

แปลงนิพจน์ Infix เป็น Postfix

ดังกล่าวแล้วว่าข้อมูลที่เราจะนำมาดำเนินการเป็นสตริง แต่เราจะมองว่าประกอบด้วย **โอเปอแรนด์ (operand)** คือ ข้อมูลที่เป็นตัวตั้งหรือตัวกระทำต่อ ซึ่งอาจจะอยู่ในรูปของตัวแปรหรือเลขจำนวนกับ**ตัวดำเนินการ (operator)** คือเครื่องหมายในการคำนวณ อีกอันหนึ่งคือวงเล็บ ข้อมูลแต่ละหน่วยที่กล่าวมานี้เรียกว่า **โทเค็น (token)** จึงมองสตริงนี้ว่าเป็นอินสตรีม (input stream) ของโทเค็น เมื่อแปลงเป็น Postfix แล้วจะได้เป็นเอาต์สตรีมของโทเค็นเช่นกัน

1 กรรมวิธีที่กล่าวในหัวข้อนี้นำมาจากหนังสือ **"โครงสร้างข้อมูล : เพื่อการออกแบบโปรแกรมคอมพิวเตอร์"** โดย ดร.สุชาย ธนวเสถียร และวิชัย จิวังกูร, บริษัท ซีเอ็ดยูเคชั่น จำกัด, (2527)

การเขียนโปรแกรมเพื่อดำเนินการในเรื่องนี้ จะต้องใช้สแต็กช่วย โดยที่จะอ่านข้อมูลจากอินสตรีมมาทีละหนึ่งโทเค็นและดำเนินการดังนี้ (ขอให้ระลึกไว้ด้วยครับว่า ตัวดำเนินการที่มีลำดับความสำคัญสูงจะอยู่ตอนต้นของเอาต์สตรีม)

1. ถ้าเป็นโอเปอแรนด์ ให้นำไปไว้ที่เอาต์สตรีม

2. ถ้าเป็นตัวดำเนินการ ให้

- ถ้าสแต็กว่างให้เก็บลงสแต็ก
- ถ้าสแต็กไม่ว่างให้เทียบตัวดำเนินการใหม่กับตัวเดิมที่อยู่ชั้นบนของสแต็ก โดย

- ถ้าตัวใหม่มีลำดับความสำคัญสูงกว่า ให้เก็บลงสแต็ก

- ถ้าตัวใหม่มีลำดับความสำคัญน้อยกว่า หรือเท่ากัน ให้ถอนตัวดำเนินการในสแต็กไปไว้ที่เอาต์สตรีม แล้วทำซ้ำจนกว่าตัวใหม่มีความสำคัญสูงกว่า หรือพบเครื่องหมายวงเล็บเปิด "(" หรือสแต็กว่าง ให้เก็บตัวใหม่ลงสแต็ก

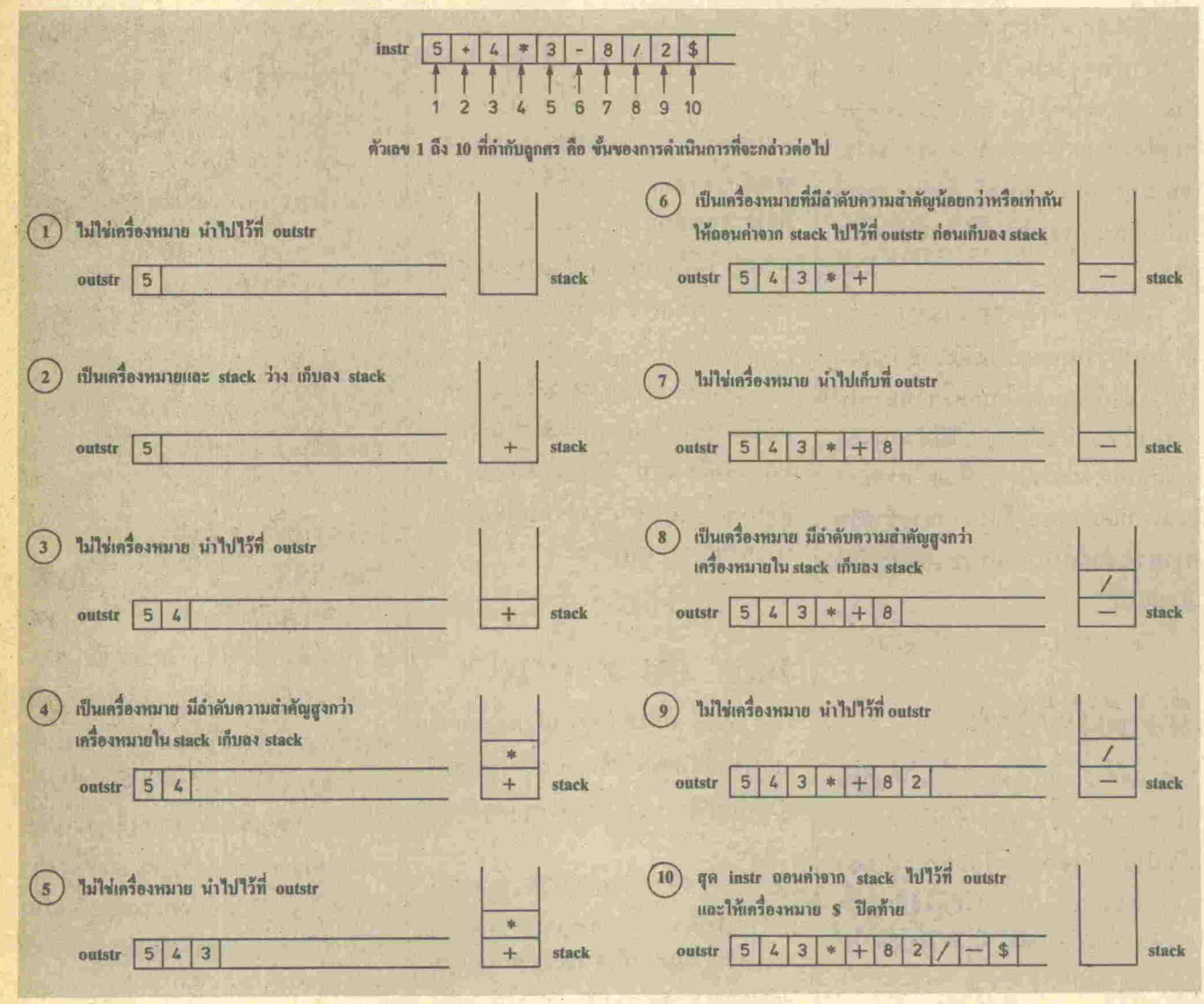
3. ถ้าเป็นเครื่องหมายวงเล็บเปิด "(" ให้เก็บลงสแต็ก

4. ถ้าเป็นเครื่องหมายวงเล็บปิด ")" ให้ถอนตัวดำเนินการไปไว้ที่เอาต์สตรีม จนกว่าจะถึงเครื่องหมายวงเล็บปิด

5. เมื่อหมดโทเค็นในอินสตรีมแล้ว ให้ถอนโทเค็นในสแต็กไปไว้ที่เอาต์สตรีม

ดังตัวอย่างการแปลงอินสตรีมเป็นเอาต์สตรีมในรูปที่ 1

แต่มีเครื่องหมายบางตัวที่เราต้องให้น้ำหนัก เมื่ออยู่ในอินสตรีมกับเมื่ออยู่ในสแต็กไม่เท่ากัน คือเครื่องหมายยกกำลัง ตามกรรมวิธีที่กล่าวมาเครื่องหมายที่มีระดับความสำคัญเท่ากัน จะดำเนินการจากซ้ายไปขวา เช่น 2+3−4 จะดำเนินการเป็น 5−4 แต่การยกกำลังจะต้องดำเนินการจากขวามาซ้าย เช่น 2^3^4 จะดำเนินการเป็น 2^81 จะทำได้โดยให้ระดับความสำคัญที่อินสตรีมสูงกว่าที่สแต็ก อีกตัวหนึ่งคือเครื่องหมายวงเล็บเปิด เมื่ออยู่ในอินสตรีมจะต้องมี



รูปที่ 1 แสดงการแปลงนิพจน์แบบ Infix ใน instr เป็นนิพจน์แบบ Postfix ใน outstr
ข้อมูลที่ให้แปลงคือ *5+4*3−8/2* เท่ากับ *13*

ลำดับความสำคัญสูงสุด เพื่อให้ได้เก็บลงสแต็กเสมอ แต่เมื่ออยู่ในสแต็กจะต้องมีลำดับความสำคัญต่ำสุด เพื่อไม่ให้ถูกถอนออกจากสแต็กเนื่องจากการเปรียบเทียบ ได้ทำเป็นตารางไว้ในรูปที่ 2

เครื่องหมาย	ลำดับความสำคัญที่อินพุต	ลำดับความสำคัญที่สแต็ก
+	1	1
−	1	1
*	2	2
/	2	2
^	4	3
(	4	0

รูปที่ 2 แสดงลำดับความสำคัญของเครื่องหมายที่อินพุตสตรีม และที่สแต็ก

การคำนวณ

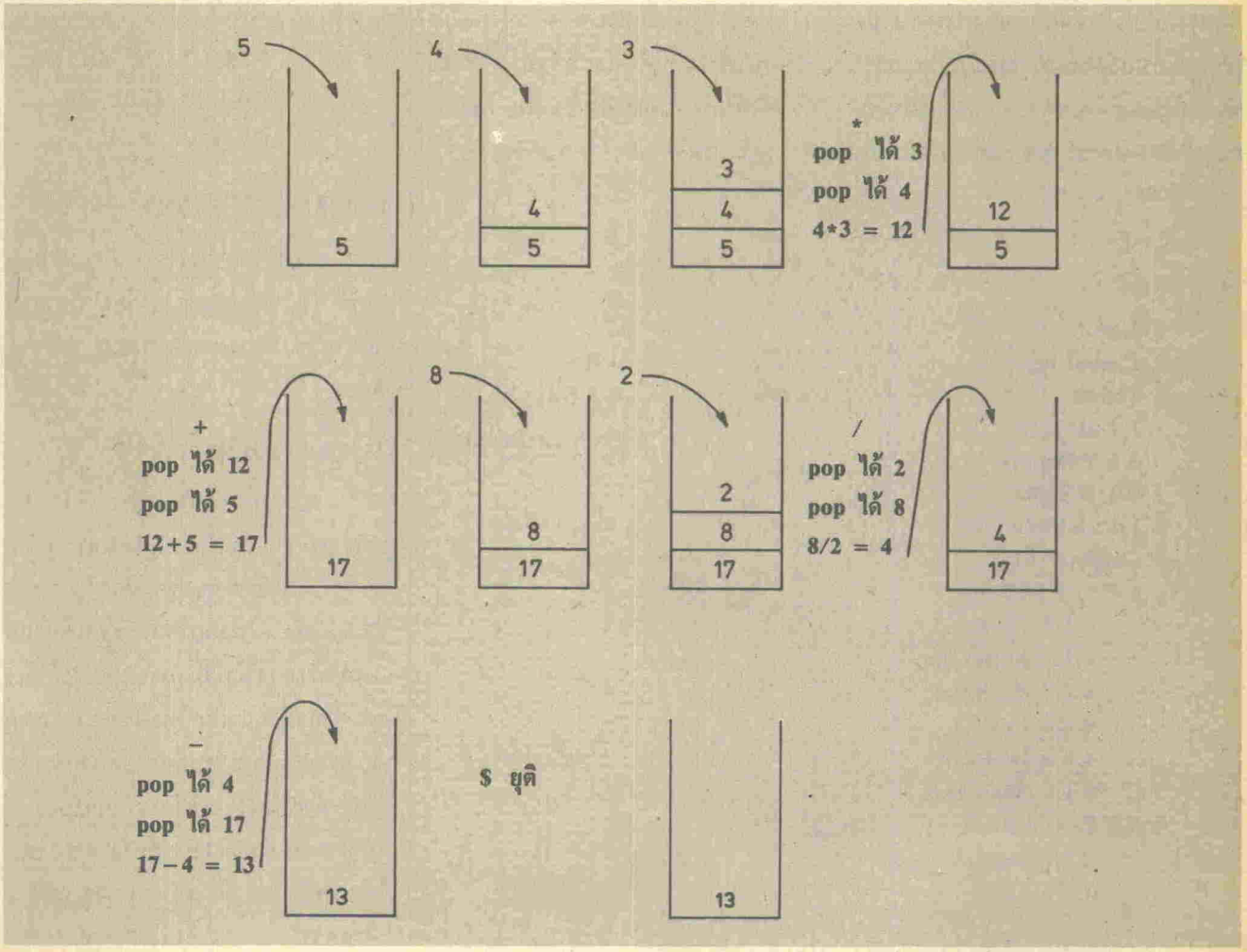
เมื่อข้อมูลอยู่ในรูปของนิพจน์แบบ Postfix ในเอาต์สตรีมแล้ว ขั้นตอนต่อไปเป็นการคำนวณหาผลลัพธ์ ซึ่งมีหลักการอยู่ว่า ให้อ่านโทเค็นไปทีละตัว จนพบตัวดำเนินการ ให้นำโทเค็นที่อยู่ข้างหน้า 2 ตัวมาเป็นโอเปอแรนด์ โดยให้ตัวหน้าเป็นตัวตั้งและตัวหลังเป็นตัวกระทำต่อ แต่ในการเขียนโปรแกรมเราจะใช้สแต็กช่วย โดยอ่านโทเค็นจากเอาต์สตรีมทีละตัวมาดำเนินการดังนี้

1. ถ้าเป็นโอเปอแรนด์ ให้เก็บลงสแต็ก

2. ถ้าเป็นตัวดำเนินการ ให้ถอนค่าจากสแต็ก 2 ค่า โดยที่ค่าแรกที่ถอนเป็นตัวหลัง ค่าที่สองเป็นตัวตั้ง นำมาดำเนินการคำนวณ ผลลัพธ์เก็บลงสแต็ก

3. ทำซ้ำจนหมดโทเค็นในเอาต์สตรีม จะได้ผลลัพธ์อยู่ในสแต็ก ดังตัวอย่างในรูปที่ 3

รูปที่ 3 แสดงการคำนวณจากนิพจน์ที่เป็น Postfix ข้อมูลที่นำมาคำนวณใช้ผลจากรูปที่ 1 คือ 5 4 3 * + 8 2 / − $ จะดำเนินการได้ดังนี้



สแต็ก

การใช้งานสแต็กในภาษาเครื่องจะมีคำสั่งในการเก็บค่าลงสแต็ก (push) และถอนค่าจากสแต็ก (pop) โดยเฉพาะซึ่งซีพียูจะดำเนินการกับสแต็กโดยตรงและโดยอัตโนมัติโดยที่ค่าที่เก็บลงสแต็กเป็นค่าแรกจะถูกถอนเป็นค่าสุดท้าย ค่าที่เก็บลงสแต็กเป็นค่าสุดท้ายจะถูกถอนเป็นค่าแรก (LIFO : last-in first-out)

สำหรับการใช้งานสแต็กในภาษาสูงเช่นในกรณีนี้ เราไม่ได้ใช้สแต็กของซีพียู หากกำหนดหน่วยความจำขึ้นหน่วยหนึ่งให้เป็นอะเรย์ แล้วนำหลักการของสแต็กคือ LIFO มาใช้ ซึ่งในการแปลงอินสตรีมให้เป็นเอาต์สตรีม ค่าที่จะเก็บลงสแต็กคืออักขระที่เป็นเครื่องหมายในการคำนวณและวงเล็บเปิด สแต็กที่ใช้ในตอนนี้จึงเป็นอะเรย์ของอักขระ ส่วนในขั้นตอนของการคำนวณค่าที่จะเก็บลงสแต็กคือเลขจำนวน และจะให้เป็นเลขจำนวนจริง สแต็กที่ใช้ในตอนนี้จึงเป็นอะเรย์ของเลขจำนวนจริง

สตรีม

คำว่า **สตรีม** เป็นคำที่ผมกำหนดเรียกเท่านั้น เพื่อให้เห็นว่าเป็นสายของโทเค็น ซึ่งอินสตรีมจะเป็นเพียงสตริงธรรมดา สำหรับเอาต์สตรีมจะให้เป็นสตริงธรรมดาก็ได้เช่นกัน แต่เมื่อเราแปลงอินสตรีมเป็นเอาต์สตรีมนั้น เราต้องตรวจโทเค็นทุกตัวว่าเป็นแบบใด (เครื่องหมาย ตัวแปร เลขจำนวน) หากเราเก็บแบบของโทเค็นกำกับไว้ด้วยจะทำให้ไม่ต้องเสียเวลาตรวจสอบอีก จึงได้ว่าเอาต์สตรีมเป็นอะเรย์ของเรคอร์ด คือ

```
OutStream =
    ARRAY[1..MAXTOK] OF
      RECORD
      typ : TypeOfToken;
      tok : String[12]
      END;
```

ซึ่ง TypeOfToken จะกำหนดให้เป็นแบบของโทเค็นดังกล่าวมา ส่วนการที่ให้ tok เป็นสตริง 12 นั้น เพราะตัวเลขที่แสดงค่าเลขจำนวนจริงจะยาวได้ไม่เกิน 11 ตัว รวมจุดอีก 1 ตัว เป็น 12 ตัว และผมมองไม่เห็นความจำเป็นใดที่จะต้องตั้งชื่อตัวแปรให้ยาวกว่า 12 อักขระ จึงใช้ค่านี้

ตัวแปร

ตัวแปรในที่นี้หมายถึง ตัวแปรที่ผู้ใช้กำหนดขึ้น ซึ่งเราจะนำมาใช้ในโปรแกรมของเราไม่ได้ และต้องถือว่าเป็นข้อมูลแบบสตริงเท่านั้น ส่วนค่าที่ผู้ใช้ให้แก่ตัวแปรเราอ่านมาในแบบสตริงของตัวเลขเช่นกัน นำมาแปลงเป็นเลขจำนวนเก็บไว้กำกับกัน แบบของข้อมูลที่เหมาะสมสำหรับตัวแปรของผู้ใช้คือแบบเรคอร์ด และควรเป็นแบบพอยน์-เตอร์เพราะไม่ทราบว่าผู้ใช้จะกำหนดตัวแปรกี่ตัว แต่เพื่อให้บทความนี้สั้นจะทำเป็นอะเรย์ของเรคอร์ด คือ

```
uservar =
    ARRAY[1..MAXVAR] OF
      RECORD
      iden : String[12];
      val  : Real;
      END;
```

เขียนโปรแกรม

เมื่อได้วิเคราะห์ระบบงานเตรียมการต่างๆไว้พร้อมแล้ว ก็เขียนโปรแกรมได้เป็นโปรแกรม MicroCal รูป A ถึง L

โปรแกรมหลัก

โปรแกรมหลักอยู่ในรูป L (อยู่หลังสุด ขอให้พลิกไปดูก่อนครับ) แต่ความจริงพระเอกของเรื่องนี้คือโปรแกรม Calculate โปรแกรมหลักจึงเพียงแต่แสดงถึงการนำโปรแกรม Calculate มาใช้เท่านั้น โดยให้ผู้ใช้ป้อนสูตรที่ต้องการคำนวณ จนกว่าจะต้องการเลิกให้พิมพ์คำว่า Quit ในการทำความเข้าใจโปรแกรม Calculate ต่อไป ขอให้ดูโครงสร้างของโปรแกรมนี้ในรูปที่ 4 ก่อนครับ

```
Calculate
1 CheckEnd
2 Token
  2.1 IsWhite
  2.2 IsDelimit
  2.3 IsAlpha
3 LocateVar
4 CalExpression
  4.1 ConToOutStr
      4.1.1 (Token)
      4.1.2 StoreLine
      4.1.3 Pushing
      4.1.4 Poping
      4.1.5 InStream
      4.1.6 StackTop
  4.2 DoCalculate
      4.2.1 Push
      4.2.2 Cal
```

รูปที่ 4
แสดงโครงสร้างโปรแกรม Calculate

การกำหนดค่า

ในรูป A ได้แสดงถึงการกำหนดค่าต่างๆ รวมทั้งตัวแปรให้เป็นไปดังที่กล่าวมา การกำหนดค่านี้ที่ถูกควรนำไปไว้ในโปรแกรม Calculate คือ ต้องอยู่หลังส่วนหัวของโปรแกรมนี้ แต่มีตัวแปร expr กับ vars ที่เมนโปรแกรมจะต้องใช้ด้วย และผมต้องการให้การกำหนดค่าอยู่ด้วยกัน จึงนำมาไว้ข้างนอก

ClearVars

ในการเรียกใช้โปรแกรม Calculate ค่าที่ต้องผ่านมาให้คือ ประโยคที่อ่านจากคีย์บอร์ดอยู่ในตัวแปร expr กับเราต้องเตรียมอะเรย์ของตัวแปรคือ vars โดยให้มีชื่อตัวแปรเป็น nil และมีค่าเป็น 0.0 ทั้งหมด ซึ่ง ClearVars เป็นผู้ดำเนินการในเรื่องนี้ (ตัวโปรแกรมอยู่ในรูป A)

CheckEnd

ผมไม่ได้กล่าวแต่ต้นว่า ในการใช้โปรแกรมนี้ได้กำหนดให้ผู้ใช้ป้อนสูตรเป็น 2 ลักษณะ คือ เมื่อต้องการให้ค่าแก่ตัวแปร จะเป็นการให้ค่าโดยตรงหรือค่าที่เกิดจากการคำนวณก็ตามให้ปิดท้ายประโยคด้วยจุด ถ้าต้องการให้ค่าด้วยและต้องการดูผลจากการคำนวณด้วยให้ลงท้าย ? ขอให้ดูผลการรันโปรแกรมนี้ในรูปที่ 5 ประกอบ

โปรแกรม CheckEnd ในรูป B จะเป็นผู้ตรวจว่า ถ้าอักขระตัวสุดท้ายเป็น ? จะให้ quest เป็นจริง คือเป็นประโยคคำถาม ถ้าเป็นจุดให้เป็นเท็จ คือเป็นประโยคธรรมดา พร้อมกันนี้จะเปลี่ยนอักขระที่ปิดท้ายประโยคเป็น $ เพื่อใช้ในการอ้างอิงต่อไป ซึ่งจะนำไปใช้ในโปรแกรมหลักของ Calculate ในรูป K แต่ถ้าผู้ใช้ไม่ปิดท้ายจะถือว่าเป็นประโยคที่ผิด ไม่ดำเนินการต่อไป

Token

ดังกล่าวแล้วว่า เรามองอินพุตสตรีมคือ ค่าที่อยู่ในตัวแปร expr เป็นสตรีมของโทเค็น ดังนั้น เราจะต้องแยกอักขระที่อยู่ในสตริงออกเป็นคำๆ คือโทเค็นหนึ่งๆ โปรแกรมที่ดำเนินการในเรื่องนี้คือ Token ในรูป C และทำไว้

A

```
PROGRAM  PlotGraphX(input,output);

TYPE   LongStr = String[255];
VAR    expr : LongStr;

CONST  MAXTOK = 50;
       MAXVAR = 50;
       SPACE = ' ';
TYPE   Str12 = String[12];
       TypeOfToken = (delimiter,variable,number);
VAR    stackC : ARRAY[0..5] OF Char;
       stackR : ARRAY[0..5] OF Real;
       outstr : ARRAY[1..MAXTOK] OF
                  RECORD
                typ : TypeOfToken;
                tok : Str12;
                  END;
       vars   : ARRAY[1..MAXVAR] OF
                  RECORD
                iden : Str12;
                val  : Real;
                  END;
     res : Real;
     i,idx,cntr : Integer;
     wrd,tok : Str12; typ : TypeOfToken;
     neg,err, quest, nocal,frstCh : Boolean;


PROCEDURE  ClearVars;
VAR    i : Integer;
BEGIN
  FOR i := 1 TO MAXVAR  DO
    BEGIN
    vars[i].iden := 'nil';
    vars[i].val  := 0.0;
    END;
END;
```

```
Running
Enter solution, quit by enter QUIT
:x = 1/3? = 0.333333
:x = x * 4? = 1.333333
:x=x*3? = 4.000000
:
:a=2.  :b=3.  :c=4.  :y=5.  :
:x = a*y^2 + b*y + c? = 69.000000
:y=-5.  :
:x = y^4 + y^3 + y^2 + y + 1? = 521.000000
:z = y^3? = -125.000000
:
:Kbyte = 1024.  :
:x = 2^10? = 1024.000000
:x = 2^32? = 4294967296.000000
:z = x/Kbyte? = 4194304.000000
:z = x/(Kbyte*Kbyte)? = 4096.000000
:
:x = 0.9^12? = 0.282430
:x = -0.9^12? = 0.282430
:x = -0.9^11? = -0.313811
:x = 2^(-2)? = 0.250000
:QUIT
>
```

รูปที่ 5 แสดงผลการรันโปรแกรม Microcal ตัวเลขที่อยู่หลังเครื่องหมายเท่ากับเป็นผลจากโปรแกรม

เป็นฟังก์ชันด้วย โดยที่เมื่อแยกคำแล้วจะให้ค่าแก่ตัวแปร tok และตัวมันเอง พร้อมกันนี้จะตรวจว่าเป็นคำประเภทใด ตัวแปร ตัวเลข หรือตัวดำเนินการให้แก่ตัวแปร typ

ตอนต้นของโปรแกรมมีการตรวจเครื่องหมายลบสำหรับค่าติดลบ เช่น −5 ซึ่งจะต้องอยู่ในวงเล็บ คือ ต้องป้อนสูตรเป็น

a = −5*20+(−7)

ในโปรแกรม Token ได้เรียกใช้โปรแกรมย่อยอีกหลายตัวคือ IsWhite (เป็นวรรคหรือขึ้นบรรทัดใหม่หรือไม่) IsDelimit (เป็นอักขระที่กำหนดใช้คั่นระหว่างโอเปอแรนด์หรือไม่ ได้แก่ เครื่องหมายในการคำนวณ เครื่องหมายวงเล็บ เครื่องหมายเท่ากับ และ $) และ IsAlpha (เป็นอักษรหรือไม่) โปรแกรมเหล่านี้อยู่ในรูป B (สำหรับ IsDigit ไม่ได้ใช้ในโปรแกรมนี้ แต่แสดงไว้เพื่อให้เห็นวิธีการ)

ขอให้เทียบโปรแกรม Token กับ ReadIden กับพวกของมันในเรื่องภาษา IQ0 จะเห็นว่าเป็นโปรแกรมที่ทำงานในเรื่องเดียวกัน แต่เขียนให้แตกต่างกัน

B

```
PROCEDURE  Calculate(expr : LongStr);

FUNCTION  IsAlpha(ch : Char) : Boolean;
BEGIN
  IsAlpha := UpCase(ch) IN ['A'..'Z']
END;

FUNCTION  IsDigit(ch : Char) : Boolean;
BEGIN
  IsDigit := ch IN ['0'..'9']
END;

FUNCTION  IsWhite(ch : Char) : Boolean;
BEGIN
  IsWhite := (ch = ' ') OR (ch = #13)
END;

FUNCTION  IsDelimit(ch : Char) : Boolean;
BEGIN
  IF  pos(ch,' +-*/^()$=') <> 0  THEN
    IsDelimit := TRUE
  ELSE
    IsDelimit := FALSE;
END;

PROCEDURE  CheckEnd(VAR quest,err : Boolean);
VAR  i : Integer;  ch : Char;
BEGIN
  err := FALSE; i := Ord(expr[0]);
  ch := expr[i];  expr[i] := '$';
  IF  ch = '?'  THEN  quest := TRUE
  ELSE IF  ch = '.'  THEN  quest := FALSE
  ELSE  err := TRUE;
END;
```

C

```
FUNCTION  Token(VAR tok : Str12; VAR typ : TypeOfToken) : Str12;
VAR  i : Integer;
BEGIN
  tok := '';
  WHILE  IsWhite(expr[cntr])  DO  cntr := cntr + 1;
  IF  frstCh AND (expr[cntr] = '-')  THEN
    BEGIN  tok := Concat(tok,'-'); cntr := cntr+1  END;
  frstCh := FALSE;
  IF  expr[cntr] = '$'  THEN  tok := '$'
  ELSE IF  IsDelimit(expr[cntr])  THEN
    BEGIN
    tok := Concat(tok,expr[cntr]);
    typ := delimiter; cntr := cntr + 1;
    IF  tok = '('  THEN  frstCh := TRUE;
    END
  ELSE
    BEGIN
    IF  IsAlpha(expr[cntr])  THEN  typ := variable
      ELSE  typ := number;
    WHILE  NOT IsDelimit(expr[cntr])  DO
      BEGIN
      tok := Concat(tok,expr[cntr]);  cntr := cntr + 1;
      END;
    END;
  Token := tok;
END;

PROCEDURE  LocateVar(wrd : Str12; VAR ind : Integer);
BEGIN
  ind := 0;
  REPEAT
  ind := ind + 1;
  IF  vars[ind].iden = 'nil'  THEN  vars[ind].iden := wrd;
  UNTIL  (wrd = vars[ind].iden) OR (ind > MAXVAR);
  IF  ind > MAXVAR  THEN  WriteLn('ERROR: can not assign var');
END;
```

LocateVar

ในเมนโปรแกรมของ Calculate จะถือว่าโทเค็นแรกเป็นตัวอักษร จึงนำไปตรวจหาว่าเป็นตัวแปรเก่าหรือไม่ ถ้าไม่ใช่จะกำหนดให้เป็นตัวแปรตัวใหม่ (เช่นเดียวกับในภาษาเบสิก) โปรแกรมที่ดำเนินการในเรื่องนี้คือ LocateVar ในรูป C โปรแกรมนี้แยกอักษรเล็กกับอักษรใหญ่คือ a กับ A เป็นคนละตัวกัน ดังนั้นตัวแปร x กับ X จึงเป็นคนละตัวกัน

CalExpression

เป็นโปรแกรมสำคัญของเรื่องนี้ จะดำเนินการคำนวณสตริงตอนที่เป็นนิพจน์แท้ๆ และได้เขียนไว้ในรูปของฟังก์ชันจึงจะส่งค่ากลับมาในชื่อของตัวมันเอง มีโปรแกรมหลักอยู่ในรูป J (ขอให้พลิกไปดูด้วยครับ)

จะดำเนินการตามขั้นตอนวิธีที่กล่าวมาคือ แปลงอินสตรีมในรูปของ Infix เป็นเอาต์สตรีมในรูปของ Postfix แล้วดำเนินการคำนวณด้วยโปรแกรมย่อยที่จะกล่าวต่อไปนี้

ConvToOutStr

จะแปลงอินสตรีมเป็นเอาต์สตรีม แต่ขอกล่าวถึงโปรแกรมย่อยที่สนับสนุนมันก่อน คือ

- **StoreLine** ในรูป D จะเก็บโทเค็นลงเอาต์สตรีม (outstr) โดยเก็บแบบของโทเค็น (ตัวแปร ตัวเลข หรือเครื่องหมาย) กำกับไว้ด้วย
- **Pushing** อยู่ในรูป D เช่นกัน จะเก็บ tok ซึ่งในกรณีนี้คือเครื่องหมายเท่านั้นลงสแต็ก stackC
- **Poping** อยู่ในรูป D เช่นกัน จะถอน tok ที่เก็บไว้ในสแต็กออก แต่ผมได้ให้ทำเกินเลยไปกว่าการถอนค่าจากสแต็กโดยปกติ คือ ให้นำไปเก็บที่เอาต์สตรีมด้วย
- **InStream** อยู่ในรูป E เป็นฟังก์ชันให้น้ำหนักของเครื่องหมายเมื่ออยู่ที่อินสตรีม
- **StackTop** อยู่ในรูป E เช่นกัน เป็นฟังก์ชัน ให้น้ำหนักของเครื่องหมายเมื่ออยู่ในสแต็ก

ตัวโปรแกรมหลักของ ConvTo OutStr ในรูป F จะดำเนินการแปลงอินสตรีมเป็นเอาต์สตรีม ตามขั้นตอนวิธีที่กล่าวไว้ในหัวข้อแปลงนิพจน์ Infix เป็น Postfix

DoCalculate

จะดำเนินการคำนวณนิพจน์ที่เป็น Postfix ซึ่งจะขอกล่าวถึงโปรแกรมย่อยที่สนับสนุนก่อนคือ

- **Push** อยู่ในรูป G จะเก็บโทเค็นซึ่งในกรณีนี้คือโอเปอแรนด์ลงสแต็ก stackR โดยที่ถ้าเป็นตัวแปรจะไปนำค่าที่เป็นเลขจำนวนจริง ซึ่งเก็บไว้ในตัวแปร vars มาเก็บลงสแต็ก ถ้าไม่ใช่ก็จะต้องเป็นสตริงของตัวเลขให้เปลี่ยนเป็นเลขจำนวน แล้วเก็บลงสแต็ก แต่ถ้าผู้ใช้ป้อนตัวเลขผิด เช่นป้อนเป็น 10,000 ซึ่งเราไม่ได้กำหนดให้ใช้ , ในการป้อนตัวเลข ให้แสดงข้อผิดพลาด

D

```
(*=========================================================*)
FUNCTION  CalExpression : Real;

  PROCEDURE  ConvToOutStr;
  VAR  ind,SP : Integer;  tem : Str12;
       tok : Str12; typ : TypeOfToken;

  PROCEDURE  StoreLine(tok : Str12; typ : TypeOfToken);
  BEGIN
    outstr[ind].typ := typ;
    outstr[ind].tok := tok;  ind := ind + 1;
  END;

  PROCEDURE  Pushing(tok : Str12);
  BEGIN
    stackC[SP] := tok;  SP := SP + 1;
  END;

  PROCEDURE  Poping(VAR tok : Str12);
  BEGIN
    SP := SP - 1;  tok := stackC[SP];
    IF  tok <> '('  THEN  StoreLine(tok,delimiter);
  END;
```

E

```
FUNCTION  InStream(ch : Char) : Integer;
BEGIN
  CASE  ch  OF
    '+' : InStream := 1;
    '-' : InStream := 1;
    '*' : InStream := 2;
    '/' : InStream := 2;
    '^' : InStream := 4;
    '(' : InStream := 4;
  END;
END;

FUNCTION  StackTop : Integer;
VAR  ch : Char;
BEGIN
  ch := stackC[SP-1];
  CASE  ch  OF
    '+' : StackTop := 1;
    '-' : StackTop := 1;
    '*' : StackTop := 2;
    '/' : StackTop := 2;
    '^' : StackTop := 3;
    '(' : StackTop := 0;
  END;
END;
```

F

```
BEGIN  { ConvToOutStr }
  ind := 1; SP := 0;  frstCh := TRUE;
  WHILE  Token(tok,typ) <> '$'  DO
    BEGIN
    IF  typ <> delimiter  THEN  StoreLine(tok,typ)
    ELSE IF  SP = 0  THEN  Pushing(tok)
    ELSE IF  tok = ')'  THEN
      BEGIN
      WHILE (tok <> '(') AND (SP > 0)  DO  Poping(tok);
      IF  SP = 0  THEN  WriteLn('ERROR: ( is missing');
      END
    ELSE
      BEGIN
      WHILE  (InStream(tok) <= StackTop) AND (SP > 0)
             AND (stackC <> '(')  DO  Poping(tem);
      Pushing(tok);
      END;
    END;  { while }
  WHILE  SP > 0  DO  Poping(tem);
  StoreLine('$',delimiter);
END;
```

● **Cal** ในรูป H จะดำเนินการคำนวณค่าที่ผ่านมาให้แก่ตัวแปร frst และ scnd ตามเครื่องหมายในการคำนวณที่ผ่านมาให้ในตัวแปร opr โดยให้ frst เป็นตัวตั้ง และ scnd เป็นตัวกระทำต่อ เช่นเป็นตัวบวก (สำหรับกรรมวิธีที่ใช้ในการยกกำลังยังไม่ดีจะต้องหากรรมวิธีที่ดีกว่านี้ครับ)

ตัวโปรแกรมหลักของ DoCalculate อยู่ในรูป I ดำเนินการคำนวณนิพจน์ที่อยู่ในเอาต์สตรีม ตามที่กล่าวในหัวข้อการคำนวณ ได้ค่าอยู่ที่ชั้นยอดของสแต็ก แต่ก็คือชั้นล่างสุดของสแต็กในขณะนั้นนั่นเอง

ตัวโปรแกรมหลักของโปรแกรม CalExpression ในรูป J จึงเพียงแต่เรียกใช้ ConvToOutStr เพื่อเปลี่ยนอินสตรีมเป็นเอาต์สตรีม เรียกใช้ DoCalculate เพื่อดำเนินการคำนวณ แล้วให้ผลในการคำนวณกับตัวมันเองส่งกลับไป

Calculate

เป็นโปรแกรมใช้งานโปรแกรม CalExpression แบบหนึ่ง คือ จะดำเนินการคำนวณสูตรซึ่งผู้ใช้ป้อนและโปรแกรมหลักของ MicroCal ผ่านค่ามาให้แก่ตัวแปร expr

ขอให้ดูในรูป K เริ่มด้วยการตรวจท้ายประโยคในตัวแปร expr ว่า ปิดท้ายด้วยเครื่องหมายคำถามหรือจุดตามที่กำหนดไว้หรือไม่ หากถูกต้องจะดำเนินการต่อไป โดยอ่านโทเค็นมา 1 ตัว ซึ่งจะต้องเป็นตัวแปร ถ้าถูกต้องจะดำเนินการต่อไปคือ กำหนดตัวแปรด้วย LocateVar แล้วอ่านโทเค็นมาอีก 1 ตัว ซึ่งจะต้องเป็นเครื่องหมายเท่ากับ หากถูกต้องจะดำเนินการคำนวณโดยเรียกใช้ CalExpression ให้ค่าแก่ตัวแปร res

ในการดำเนินการคำนวณนี้ เราจะทราบด้วยว่ามีการคำนวณจริงๆหรือ

G

```
(*.....................................................*)
  PROCEDURE  DoCalculate;
  VAR  ind,SP : Integer;  frst,scnd : Real;
       tok : Str12; typ : TypeOfToken;

  PROCEDURE  Push(wrd : Str12; typ : TypeOfToken);
  VAR  ind,code : Integer;  value : Real;
  BEGIN
    ind := 1;
    IF  typ = variable  THEN
      BEGIN
      WHILE  (wrd <> vars[ind].iden) AND (ind < MAXVAR)  DO
        ind := ind + 1;
      IF  ind <= MAXVAR  THEN
        value := vars[ind].val
      ELSE
        BEGIN WriteLn('ERROR; ',wrd,' is not an assign var');
              value := 1;  END;
      END
    ELSE
      BEGIN
      Val(wrd,value,code);
      IF code <> 0  THEN
        BEGIN WriteLn('ERROR: ',wrd,' is not number');
              value := 1  END;
      END;
    stackR[SP] := value;
    SP := SP + 1;
  END;
```

H

```
  PROCEDURE  Cal(VAR frst,scnd : Real; opr : Char);
VAR  mid : Real;
  BEGIN
    CASE  opr  OF
    '+' : frst := frst + scnd;
    '-' : frst := frst - scnd;
    '*' : frst := frst * scnd;
    '/' : if scnd <> 0.0 then frst := frst / scnd;
    '^' : begin
            neg := FALSE;
            if frst < 0  then
              begin  frst := abs(frst); neg := TRUE end;
            if (frst <> 0)  then
              begin
              mid := scnd * Ln(frst);
              if mid < -30.0 then mid := -29.9999999;
              frst := Exp(mid)
              end;
            if  neg and odd(round(scnd))  then
              frst := frst - frst*2;
          end;
    END;  { case }
  END;
```

I

```
BEGIN   { DoCalculate }
ind := 1;  SP := 0; nocal := FALSE;
IF  outstr[ind].tok <> '$'  THEN
  WHILE  outstr[ind].tok <> '$'  DO
    BEGIN
    tok := outstr[ind].tok;  typ := outstr[ind].typ;
    IF  (typ = variable) OR (typ = number)  THEN
      Push(tok,typ)
    ELSE
      BEGIN
      SP := SP - 1; scnd := stackR[SP];
      SP := SP - 1; frst := stackR[SP];
      cal(frst,scnd,tok);
      stackR[SP] := frst;  SP := SP + 1;
      END;
    ind := Ind + 1;
    END
  ELSE
    nocal := TRUE;
END;
```

ไม่ หากไม่มีการคำนวณคือ nocal เป็นจริง เช่นเมื่อผู้ใช้ป้อนสูตรว่า x = ? จะให้ res มีค่าเป็นค่าของตัวแปรตัวนั้น เช่นจากคำถามนี้คือค่าของ x หากมีการคำนวณก็จะให้ตัวแปรมีค่าเป็น res ต่อไป จะตรวจว่าประโยคที่ผู้ใช้ป้อนเป็นคำถามหรือไม่ ถ้าเป็นคำถามจะแสดงค่าของ res บนจอภาพ ตอนท้ายของโปรแกรมนี้เป็นการแสดงข้อความบ่งถึงข้อผิดพลาดหากผู้ใช้ป้อนสูตรผิดไปจากที่กำหนด

โปรแกรม Calculate นี้อยู่ในวงรอบการทำงานของโปรแกรม Micro Cal รูป L ซึ่งได้กล่าวแต่ตอนต้นแล้ว

สเปรดชีต

การใช้งานโปรแกรม MicroCal ดังแสดงในรูปที่ 5 หากต้องการทำเป็นสเปรดชีตอย่างง่ายๆ เช่น เป็นสเปรดชีตขนาด 10×10 เซลล์ ในแต่ละเซลล์ให้เก็บโทเค็นได้ 80 อักขระ (ของเดิมกำหนดไว้ 255 อักขระดูจะมากเกินไป) จะกำหนดตัวแปร vars ได้เป็น

VARS : ARRAY[A..J, 1..10] OF String80;

และให้ผู้ใช้กำหนดตัวแปรตายตัว คือ A1, A2 จนถึง J10 จะหมายถึงแต่ละเซลล์ ในการป้อนสูตรอาจให้ป้อนเฉพาะนิพจน์ตามแบบสเปรดชีตทั้งหลาย ซึ่งจะต้องมีกรรมวิธีในการอ้างถึงเซลล์แต่ละเซลล์ หรืออาจให้ป้อนดังที่ดำเนินการมา เช่น

A1 = apple.
A2 = 30.
A3 = 15.50.
A4 = A2*A3?

โดยที่ผู้ใช้อาจหมายความว่า แอปเปิ้ล จำนวน 30 ผล ราคาผลละ 15.50 บาท รวมเท่าไร ซึ่งเราจะต้องเปลี่ยนรูปแบบของโปรแกรมใหม่ คือแทนที่จะเก็บค่าหรือผลของการคำนวณไว้ในตัวแปรเปลี่ยนเป็นเก็บนิพจน์ (โดยถือว่าโทเค็นที่อยู่หลังเครื่องหมายเท่ากับเป็นนิพจน์) ไว้ในเซลล์ โดยยังคงเก็บอยู่ในรูปของสตริง เมื่อถามถึงค่าในเซลล์หนึ่งๆ เช่นอาจถามว่า A3 = ? หรือตามตัวอย่าง ก็จะนำค่าออกแสดง หากมีการคำนวณก็จะดำเนินการคำนวณก่อนก็จะได้สเปรดชีตอย่างง่ายๆ (การออกแบบจอภาพเป็นอีกเรื่องหนึ่ง)

ขอให้ทดลองทำดูครับ แต่ต้องเปลี่ยนโปรแกรมหลายตอน เช่นใน Token จะต้องถือว่าตัวแปรจะต้องประกอบด้วยตัวอักษร A ถึง J หนึ่งตัวกับตัวเลข 1 ถึง 10 หนึ่งหรือสองตัว นำไปทำเป็นอินเด็กซ์ เช่นให้ตัวอักษรแก่ตัวแปร row ตัวเลขแก่ตัวแปร col จะอ้างถึงเซลล์แต่ละเซลล์ ได้เป็น

vars[row,col]

ซึ่งในแต่ละเซลล์จะเก็บนิพจน์ดังกล่าว

J

```
(*.....................................................*)

BEGIN  { CalExpression }
  ConvToOutStr;
  DoCalCulate;
  CalExpression := StackR[0];
END;
```

K

```
(*=====================================================*)

BEGIN  {Calculate}
  cntr := 1;
  CheckEnd(quest,err);
  IF  err  THEN  WriteLn('NO ? or . ')
  ELSE
    BEGIN
    wrd := Token(tok,typ);
    IF  typ = variable  THEN
      BEGIN
      LocateVar(tok,idx);
      IF  Token(tok,typ) = '='  THEN
        BEGIN
        res := CalExpression;
        IF nocal  THEN  res := vars[idx].val
        ELSE  vars[idx].val := res;
        IF  quest  THEN
          WriteLn(' = ',res:8:6)
        ELSE
          Write('  ');
        END
      ELSE
        BEGIN  WriteLn('ERROR: no = '); err := FALSE  END;
      END
    ELSE
      BEGIN WriteLn(wrd,'must be identifier'); err := FALSE END;
    END;
END;
```

L

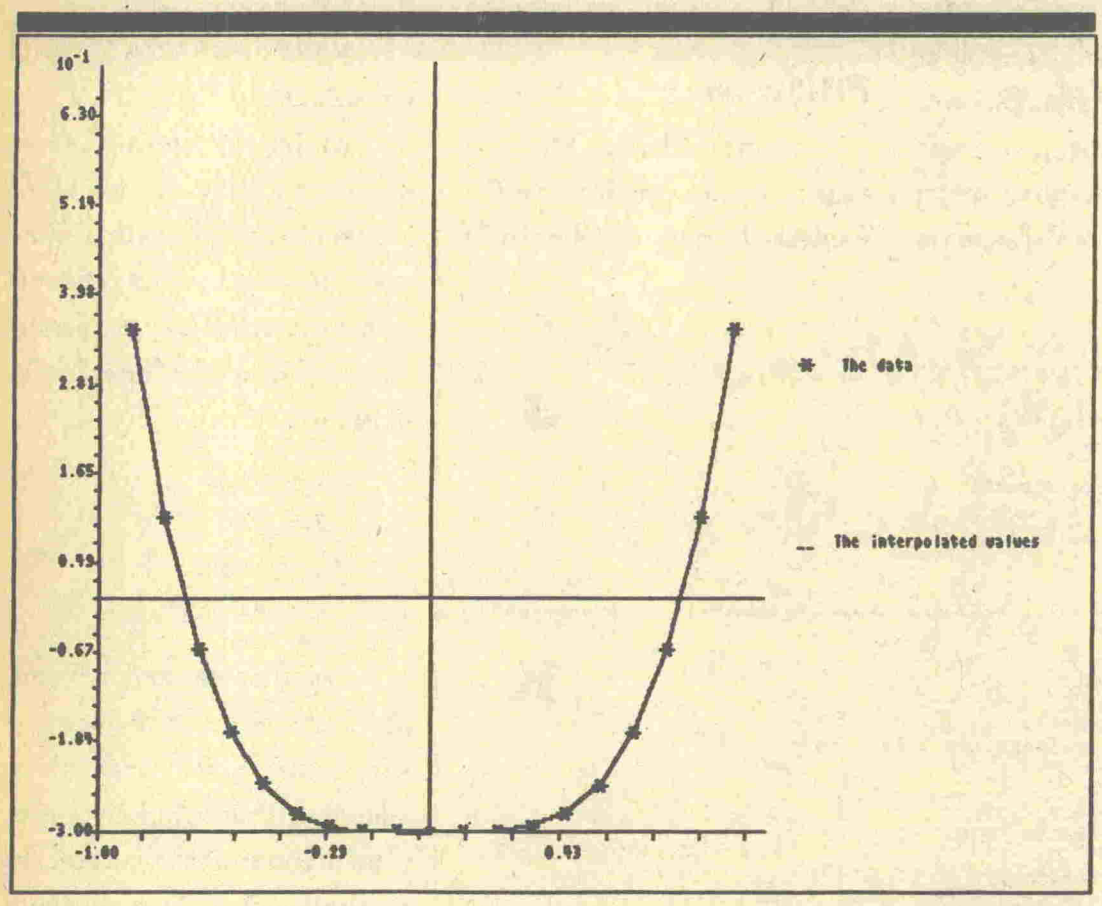
```
(*=====================================================*)
{ MAIN PROGRAM OF MicroCal }

{

BEGIN
ClearVars;
WriteLn('Enter solution, quit by enter QUIT');
Write(':'); Read(expr);
WHILE  expr <> 'QUIT'  DO
  BEGIN
  IF  expr = '' THEN
    WriteLn
  ELSE
    Calculate(expr);
  Write(':'); Read(expr);
  END;
END.
```

โปรแกรมช่วยการศึกษา

เราสามารถนำโปรแกรม Micro Cal มาดัดแปลงเพื่อให้แสดงกราฟของฟังก์ชันทางพีชคณิตได้ดังที่แสดงเป็นตัวอย่างในรูปที่ 6 ขอให้ป้อนโพรซีเยอร์ PlotGraph ในรูปที่ 7 เก็บไว้ในไฟล์



รูปที่ 6 กราฟของ $y = x^4 - 0.3$

```
procedure PlotGraph(A : PlotArray; N : Integer);
begin
  InitGraphic;                          { Initialize the graphics system }

  DefineWindow(1, 0, 0, XMaxGlb, YMaxGlb); { Define both windows as }
  DefineWindow(2, 0, 0, XMaxGlb, YMaxGlb); {   whole screen }
  DefineWorld(1, 0, 0, 1000, 1000);     { Give a world to the screen }

  DefineHeader(2, 'A spline interpolation'); { Window where curves will go }
  SetHeaderOn;

  FindWorld(2, A, N, 1, 1);          { Make world 2 the right size }
  SelectWindow(2);                      { Select it and draw border }
  DrawBorder;

  SetLineStyle(0);
  DrawAxis(5, 5, 3, 5, 25, 10, 0, 0, false);
  DrawPolygon(A, 2, N - 1, 7, 2, 0);    { Don't draw the endpoints }

  SelectWorld(1);                       { Select outside window }
  SelectWindow(1);

  DrawTextW(730, 400, 1, ^['7@2   The data');     { Print legend }
  DrawTextW(730, 600, 1, '__  The interpolated values');

  repeat until KeyPressed;              { Wait until a key is pressed }
  LeaveGraphic;                         { Leave the graphics system }
end; { PlotGraph }
```

รูปที่ 7 โพรซีเยอร์ PlotGraph

ชื่อ GRP0.SBR โปรแกรมย่อยนี้ผมนำมาจาก Turbo Pascal Graphix ToolBox โดยตัดทอนบางส่วนออก

โปรแกรม PlotGraph จะพล็อตจุด ซึ่งให้ค่าโคออร์ดิเนต x และ y (หลายๆจุด) ไว้ในอะเรย์สองมิติ ซึ่งกำหนดแบบไว้ในชื่อ PlotArray คือใน ToolBox ได้กำหนดไว้เป็น

```
TYPE PlotArray = ARRAY [1..100,1..2] OF Real;
```

เมื่อป้อนและเก็บโปรแกรม Plot Graph แล้วให้ copy โปรแกรม Micro Cal ให้แก่ไฟล์ชื่อใหม่ เช่น PLOTGRAP.PAS (ควรจะทำการ copy ก่อนป้อนโปรแกรม PlotGraph) เสร็จแล้วให้ดำเนินการอีดิตโปรแกรมในไฟล์ PLOTGRAP.PAS โดยเปลี่ยนชื่อโปรแกรมเป็น PlotGraphX (หรือชื่ออื่นตามที่เห็นควร) แล้วดำเนินการดังต่อไปนี้คือ ลบตัวเมนโปรแกรมของโปรแกรม MicroCal เดิม (รูป L) ออกทั้งหมด และป้อนโปรแกรมในรูปที่ 8 แทนที่

จะเห็นว่าในส่วนของโปรแกรมที่ป้อนใหม่นี้ได้ include ไฟล์เข้ามามากมาย ไฟล์เหล่านี้อยู่ในแผ่นดิสค์ของ ToolBox ดังกล่าว ดังนั้น ในการรันโปรแกรมนี้คุณจะต้องมีดิสค์แผ่นหนึ่งและรวมไฟล์ PLOTGRAP.PAS กับ GRP0.SBR ไว้ในแผ่นนี้ด้วย

ตัวเมนโปรแกรมใหม่นี้คงดำเนินการคล้ายเดิมคือ ให้ผู้ใช้ป้อนสูตรตามแบบที่ได้แสดงเป็นตัวอย่าง เสร็จแล้วส่งไปคำนวณด้วยการเรียกใช้ CalPoint ซึ่งจะทำการคำนวณหาค่าของ y ตามค่า x ตั้งแต่ -1.0 ถึง 1.0 โดยเพิ่มค่าครั้งละ 0.1 ด้วยการเรียกใช้ Calculate อีกทอดหนึ่ง เมื่อคำนวณเสร็จจะส่งค่ากลับมาให้แก่อะเรย์ points ซึ่งเป็นอะเรย์แบบ PlotArray

เมื่อได้ค่าจุด x และ y จำนวน 21 จุดมาแล้ว จะทำการพล็อตกราฟโดยเรียกใช้ PlotGraph ซึ่งจะทำการพล็อตจุดต่างๆด้วยเครื่องหมาย * แล้วจะต่อเชื่อมจุดต่างๆเหล่านี้ด้วยเส้นตรง แต่เพราะความถี่ของจุด ทำให้มองเห็นเป็นเส้นโค้งได้ดังที่เห็นในภาพประกอบ

เนื่องจากใน CalPoint ดำเนินการกับตัวแปรเพียง 2 ตัวเท่านั้น คือ x กับ y จึงขอให้แก้การกำหนดค่าคงตัว MAXVAR (ในรูป A) เป็นให้เท่ากับ 2 เพื่อไม่ให้ผู้ใช้ป้อนตัวแปรมากกว่านี้

โปรแกรมนี้ยังไม่สมบูรณ์ สามารถดัดแปลงเสริมแต่งได้อีกมากมาย ขอให้ทดลองดำเนินการครับ และจากบทความอื่นในฉบับเดียวกันนี้ คาดว่าคงจะได้แนวความคิดอีกมาก

```
(*-----------------------------------------------------------*)
{ start of program PlotGraph }

{$I typedef.sys}                    { These files must be }
{$I graphix.sys}                    { included in this order }
{$I kernel.sys}
{$I windows.sys}
{$I findwrld.hgh}
{$I axis.hgh}
{$I polygon.hgh}
{$I grp0.sbr }


{ type  PlotArray = array[1..100,1..2] of real;}
CONST  XX = 1;  YY = 2;  MAXPOINT = 21;
VAR    points : PlotArray;

PROCEDURE  CalPoint(epr : LongStr; VAR pnt : PlotArray);
VAR  i : Integer;
     dx : Real;
BEGIN
  vars[XX].iden := 'x';  vars[YY].iden := 'y';
  dx := -1.0;
  FOR  i := 1 TO MAXPOINT  DO
    BEGIN
    pnt[i,XX] := dx;  vars[XX].val := dx;
    Calculate(epr);
    pnt[i,YY] := vars[YY].val;
    WriteLn(vars[XX].val:12:6, vars[YY].val:12:6);
    dx := dx + 0.1;
    END;
END;


BEGIN  { MAIN PlotGraphX }

WriteLn('This program will plot graph of the required function');
WriteLn('Enter the equation in the form such as');
WriteLn('  y = x^2 + 4*x - 5. ');
WriteLn('Quit by enter QUIT');
Write(': ');  ReadLn(expr);
WHILE  expr <> 'QUIT'  DO
  BEGIN
  err := FALSE;
  CalPoint(expr,points); Read;
  IF  NOT err  THEN
    PlotGraph(points,MAXPOINT)
  ELSE
    Write('ERROR: in ');
  WriteLn(expr);
  Write('Enter equation or QUIT ');
  Write(': ');  ReadLn(expr);
  END;

END.
```

รูปที่ 8 โปรแกรมส่วนที่ต้องนำไปแทนเมนโปรแกรมในรูป L